ENGR 102 – Fall 2023
Exam 2 Practice Problems

Short answer problems for studying:

These review questions are to see how well you understand the concepts. There won't be any short answer problems like this, but if you can answer them, you have a good understanding of the material. If you struggle with any of these questions, keep studying that topic!

1. List 5 good coding practices that have been mentioned in this course.

2. In the following dictionary, identify the keys and values. Write the command to add the key `'Orange'` with value `1` to the dictionary. Write code to loop through the dictionary and check for a particular key or value.
```
mydict = {'Apple' : 3, 'Pear' : 5, 'Banana' : 2}
```

3. Explain how dictionaries are different from lists, and how lists are different from tuples.

4. Identify and explain 3 ways strings are similar to lists.

5. List all of the data types we have seen in this course and provide an example of each. Identify which ones are mutable and which are immutable.

6. In your own words, explain the difference between the top-down and bottom-up design methods. Name at least one advantage and one disadvantage for each.

7. Describe the various components of a hierarchy. How can we use one in program design?

8. Does it matter if we use the same variable names inside a user defined function and the main program? Why (not)?

9. Does it matter where in our code we write function definitions?

10. Can functions access main memory? How do we pass information into a function? Can functions return multiple values?

11. What type of coding error do you hate the most and why?

12. When you are trying to fix your code, what are some alternatives to using a debugger?

13. When would it be best to use the `myfile = open(…)` and `myfile.close()` commands for opening files as opposed to the `with open(…) as myfile:` command?

14. When opening files, what is the difference between the designators `r`, `w`, `r+`, and `a`?

15. What are the differences between `myfile.read()`, `myfile.readline()`, `myfile.readlines()`, and `list(myfile)`?

16. What command(s) are used to remove the leading and trailing whitespace in a string? To separate a string into a list of its components with a specified delimiter? Write an example using both.

17. Write the command to print the value of `pi` to 4 decimal places to the screen. Don't forget to import `pi` from the `math` module!

18. Write the command to import the `coolfunc()` function from the `neatomod` module in the `funpack` package and rename it to `coolf`. Now write the command to import all functions from the `neatomod` module. What command can we use to obtain a list of all functions inside the module?

19. What are the differences between `np.arange()`, `np.array()`, and `np.linspace()`? How many rows and columns will the matrix `np.arange(15).reshape(5, 3)` have?

20. What are the differences between `plt.plot()`, `plt.bar()`, `plt.hist()`, and `plt.scatter()`? How do you set the color of a line? The marker shapes? The axis labels? The title? The legend?

21. Why is it important to include docstrings when writing functions?

Autograded style problems:

For the following problems, write the output to the code. If there is an error, explain the error. Don't forget `[](){}` and/or `,` as needed.

22.
```
vector1 = (1, 2, 3)
vector2 = (2, 3, 4)
dotp = vector1 * vector2
print(f"The dot product is {dotp}")
```

23.
```
mystr = "Howdy"
mylist = [2, 0, 2, 0]
mytuple = (mystr, mylist)
mylist[3] = 7
print(mytuple)
```

24.
```
def plus1_3(x):
    return x + 1, x + 3
print(plus1_3(2)[0])
```

25.
```
def myfun():
    '''This function prints a message.'''
    print("Gig 'em Aggies!")
help(myfun)
```

26.
```
def myfunction(a, b=5, c):
    return a * b * c
print(myfunction(1, 2, 3))
```

27.
```
def add1(x):
    if x + 1 < 10:
        return x + 1
    else:
        return 'Too big!'
print(add1(1), add1(27))
```

28.
```
x = 4
y = timestwo(x)
def timestwo(x):
    return x * 2
print(y)
```

29.
```
mydict = {'Ann' : 18, 'Bob' : 20, 'Charlie' : 19}
if 'Joe' in mydict:
    print("Joe is here")
elif 'Ann' in mydict:
    print("Hi Ann")
else:
    print("Anyone?")
```

30.
```
mydict = {}
mylist = [1, 2, 3, 4, 5]
mydict['Length'] = len(mylist)
mydict['Max'] = max(mylist)
mydict['Min'] = min(mylist)
mydict['Crazy'] = mylist[1] * mylist[3] - mylist[-1]
for key in mydict:
    print(f"{key}: {mydict[key]}")
```

31.
```
mystr = "Howdy! Welcome to Texas A&M Engineering!"
mylist = mystr.split()
newstr = ''  # empty string
for i in range(3, len(mylist)):
    newstr += mylist[i] + ' '
print(mylist[0][:5] + ' ' + newstr[:-2] + ' students!')
```

32.
```
mylist = [5, 3, 7, 9, 1, 2]
mylist.pop()
mylist.insert(2, 4)
mylist.sort()
for num in mylist:
    print(num, end = ' ')
```

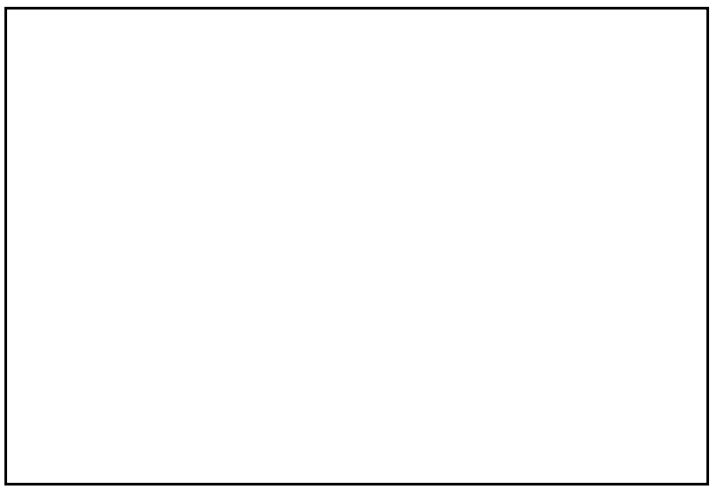
33.
```
number = input("Enter a number: ")  # assume the user enters 5.0
try:
    x = int(number)
    print(f"x is the integer {x}")
except:
    x = float(number)
    print(f"x is the float {x}")
```

34.
```
import numpy as np
mygrid = np.arange(9).reshape(3, 3)
print(mygrid[0])
print(mygrid[2][2])
```

35.
```
import numpy as np
x = np.linspace(1.0, 10.0, 10)
y = x ** 2 - 1
with open('zfile.txt', 'w') as zfile:
    zfile.write('x\ty\n')
    for i in range(len(x)):
        mystr = str(x[i]) + '\t' + str(y[i])
        zfile.write(mystr + '\n')
with open('zfile.txt') as myfile:
    all_of_it = myfile.read().split('\t')
output = ','.join(all_of_it)
print(output)
```

36. Using the box below, draw the plot produced by the following code:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(-2, 2, 25)
y1 = x
y2 = x ** 2
plt.plot(x, y1, 'g', linewidth = 3)
plt.plot(x, y2, 'k', marker = 'o', markerfacecolor = 'b')
plt.axis([-2, 2, -2, 4])
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Plots for 2 polynomials')
plt.legend(['straight', 'curved'], loc = 'upper center')
plt.show()
```



Code writing problems:

37. Assume a function `isprime()` is available for you to use in a module called `ENGR102` which determines whether or not a number is a prime number. The function `isprime()` takes in as a parameter a single integer, and returns either `True` or `False`. Write a Python program that takes as input from the user two integers. If the user gives bad input, continue to prompt them to try again until they enter two integers. Then, test only the odd numbers between and including those two numbers, to check if they are prime using the `isprime()` function. Have your program print a list of the prime numbers found. If no prime numbers were found, have your program print a message stating that. Start your code with: `from ENGR102 import isprime`. You do not have to write the function `isprime()`, you only need to call it.

Example output (bad input):
```
Enter an integer: 0.5
Bad input! Try again: 1
Enter another integer: 1.5
Bad input! Try again: 5
Primes: [2, 3, 5]
```

Example output (good input):
```
Enter an integer: 8
Enter another integer: 10
No primes found!
```

38. A text file named `data.dat` is stored on a computer's hard drive. In a text editor the file displays the snippet below. The entire file is over one hundred lines long.

Contents of `data.dat`:
```
# Created on November 5, 2023
# Time,Temperature,Windspeed
# (min),(deg F),(knots)
0,33.47,1.27
5,32.59,1.95
10,33.62,0.76
15,33.79,1.12
...
```

Write a Python program that reads the contents of this file, assigns the header lines to a variable that is a list of strings, and assigns the data to a variable that is a floating point NumPy array that contains all of the data in the file. (The data should be in an n x 3 array.) Then, find and print the minimum temperature and maximum windspeed recorded in the file. Write this code using standard file I/O commands; do not use the csv module or any other csv reader that you may know of in some Python package.

Example output (using entire file):
```
Minimum temperature is 31.28 F
Maximum windspeed is 7.82 knots
```

39. Write a Python function that takes in as parameters two numpy arrays A and B, checks that the inner dimensions of A and B match, and if they do calculate and return C = AB (matrix multiplication (not by-element)). If they don't match, have your function return an empty numpy array. The inner dimensions of two matrices multiplied together are the two interior numbers. For matrices with dimensions 2x3 and 3x4, the inner dimensions match (both 3). For matrices with dimensions 3x2 and 3x4, the inner dimensions do not match (2 and 3).

    Examples:
    Array A: `[[0 1 2] [3 4 5]]`
    Array B: `[[ 0  1  2  3] [ 4  5  6  7] [ 8  9 10 11]]`
    Function returns `[[20 23 26 29] [56 68 80 92]]`

    Array A: `[[0 1] [2 3]]`
    Array B: `[[ 0  1  2  3] [ 4  5  6  7] [ 8  9 10 11]]`
    Function returns `[]`

40. An Armstrong number is a positive integer, where the sum of its digits, each raised to the power of the number of digits, is equal to the integer itself. For example, 371 is an Armstrong number as `(3 ** 3) + (7 ** 3) + (1 ** 3) = 371`. Write a Python function called `Armstrong_number` that takes in as parameters two positive integers and returns a list containing all the Armstrong numbers between and including these two integers. Next, write main code that takes as input from the user two integers. If the user gives bad input, continue to prompt them to try again until they enter two integers that are both positive values. Then, call your function and print the resulting list.

Example output (bad input):
```
Enter an integer: -1
Need a positive integer: 0.5
Bad input! Try again: 4
Enter another integer: 2.7
Bad input! Try again: -27
Need a positive integer: 27
Armstrong numbers: [4, 5, 6, 7, 8, 9]
```

Example output (good input):
```
Enter an integer: 100
Enter another integer: 400
Armstrong numbers: [153, 370, 371]
```

41. A *perfect number* is a positive integer greater than 1 that is equal to the sum of its proper divisors. The smallest perfect number is 6 since the sum of the proper divisors for 6 (1, 2, and 3) equals 6. The integer 28 is also a perfect number since the divisors for 28 are 1, 2, 4, 7, and 14 and the sum of these divisors equals 28. Write a Python function that takes as input a positive integer greater than or equal to 1, and returns `True` if the number is a perfect number or `False` if it is not. Next, write main code that takes as input from the user one integer greater than or equal to 1. If the user gives bad input, continue to prompt them to try again until they enter a positive integer. Then, call your function and print if the number is a perfect number (or not).

    Hint: The proper divisors of a positive integer N are those numbers, other than N itself, that divide N

without remainder. For N > 1 they will always include 1, but for N == 1 there are no proper divisors. The proper divisors of 6 are 1, 2, and 3. The proper divisors of 100 are 1, 2, 4, 5, 10, 20, 25, and 50.

Example output (bad input):
```
Enter an integer: -5
Need a positive integer: 0.5
Bad input! Try again: a
Bad input! Try again: 28
28 is a perfect number
```

Example output (good input):
```
Enter an integer: 1
1 is not a perfect number
```

Example output (good input):
```
Enter an integer: 6
6 is a perfect number
```

42. A file named `item_cost.dat` is stored on a computer's hard drive. In a text editor the file displays the first few lines of the file below. The rest of the file is not shown. Write a Python program that reads the contents of this file, determines the most expensive item and prints the information about that item and its cost to the screen using the format shown below. Write your code using standard file I/O commands; *do not use* the csv module or any csv reader available in some Python package. *Do not use* any built-in sorting functions. You may assume that all costs are unique (no two items cost the same).

Contents of `item_cost.dat`:
```
Item,Cost ($)
Item1,3.75
Item2,3.50
Item3,2.75
Item4,4.50
...
```

Example output:
```
The most expensive item is Item57 and costs $5.26
```

43. Write a Python program that will take as input from the user grades from a recent ENGR 102 quiz, save it to a file named `grades.txt`, and print the average quiz grade to the screen. Have your program prompt the user for the number of students first, and then take as input that many names and their corresponding scores. You may assume the user provides valid input. Write your code using standard file I/O commands. Use the format shown below for your output file, which includes a header line and aligned columns.

Example output:
```
Enter the number of students: 5
Enter the next name and score: Amari 85
...
The average grade for this quiz is 81.4
```

Example `grades.txt` file:
```
Name     Score
Amari    85.0
Blake    76.5
Cameron  89.0
Dylan    59.5
Emerson  97.0
...
```